DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving

基本信息

Source: OSDI 2024

Authors:

Yinmin Zhong[1] Shengyu Liu[1] Junda Chen[3] Jianbo Hu[1] Yibo Zhu[2] Xuanzhe Liu[1] Xin Jin[1] Hao Zhang[3]

[1]School of Computer Science, Peking University

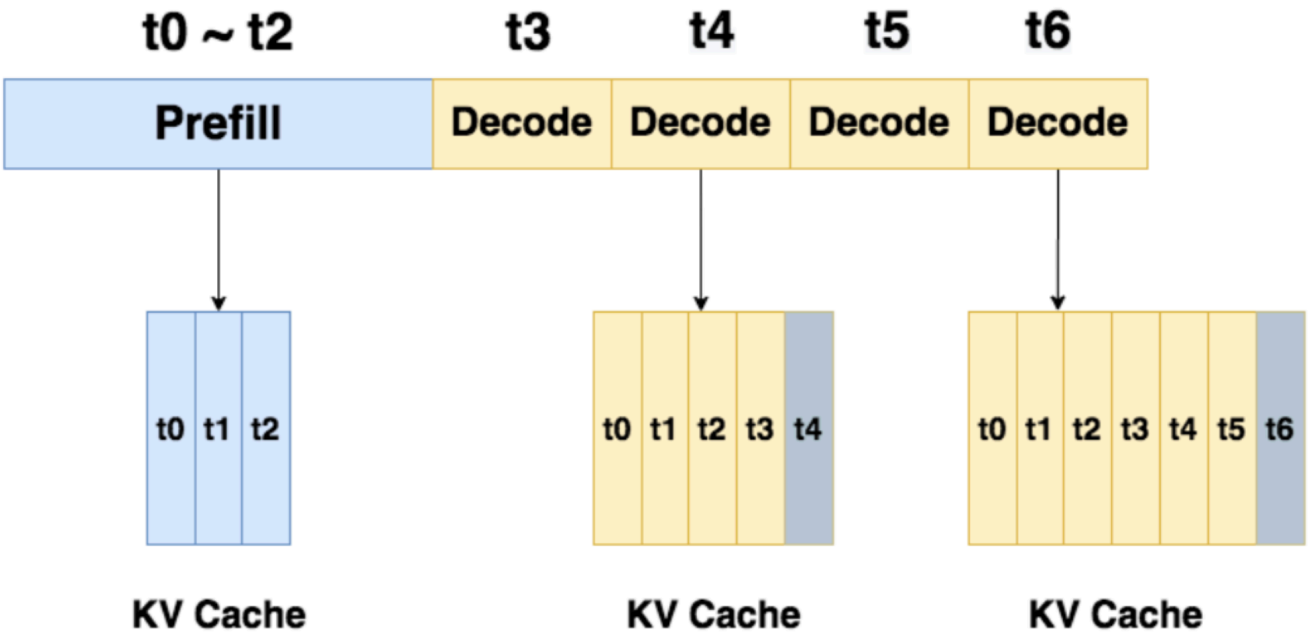
[2]StepFun

[3]UC San Diego

LLM Inference

两阶段

- **prefill阶段**：把用户 prompt输入模型做forward计算。**prefill阶段结束后，模型产出第一个token**（例如图中t3）
- **decode阶段**：**一个token一个token地产出response**。（例如图中逐一产出t4，t5，t6）



两指标

- **TTFT（Time To First Token）**：表示生成第1个token所用的时间，这是prefill阶段的重要评估指标之一

- **TPOT（Time Per Output Token）**：产出每一个response token所用的时间，这是decode阶段的重要评估指标之一

LLM**推理系统目标**

平衡 TTFT 和 TPOT，最大化 per-GPU goodput（这里的 goodput 定义为the maximum request rate that can be served adhering to the SLO attainment goal for each GPU provisioned）

Motivation

Colocate prefill and decoding

现有的 LLM 服务系统通常将两个阶段并置在 GPU 上，并通过跨请求批处理预填充和解码步骤来最大化整体系统吞吐量（所有用户和请求每秒生成的令牌）

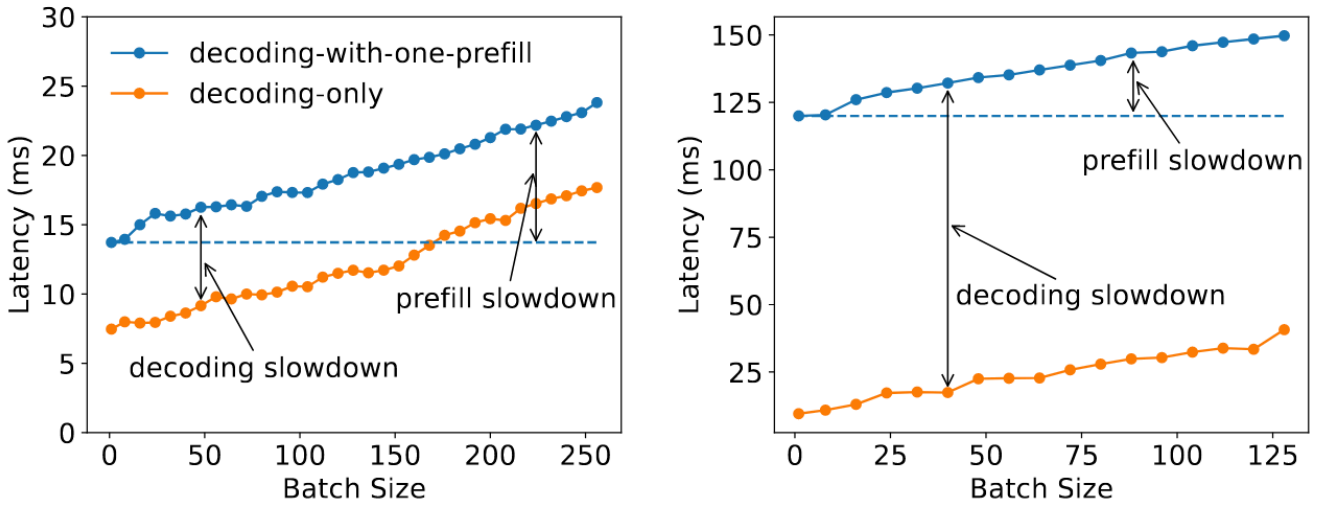
continuous batching

当前新请求到达时，先中断decode，对新请求进行 prefill，完成后再进行 decode。

将**新请求的prefill阶段**与**正在进行中的decoding阶段**合并处理

导致 trade-offs between TTFT and TPOT

Prefill-decoding interference



(a) Input length = 128 (b) Input length = 1024

Figure 2: Batch execution time when serving a 13B LLM as batch size increases. Compared between a decoding-only batch and the batch adding one more prefill job.

TTFT 和 TPOT 显着增加

- 批次中的decoding任务必须等待更长的prefill作业完成，从而延长了TPOT
- 随着prefill时间的延长，decoding slowdown会加剧，如图 2(b) 所示。

- 将decoding作业添加到prefill还会增加完成预填充任务的时间，特别是当 GPU 已满负荷时

Opportunities：disaggregate the prefill and decoding phases

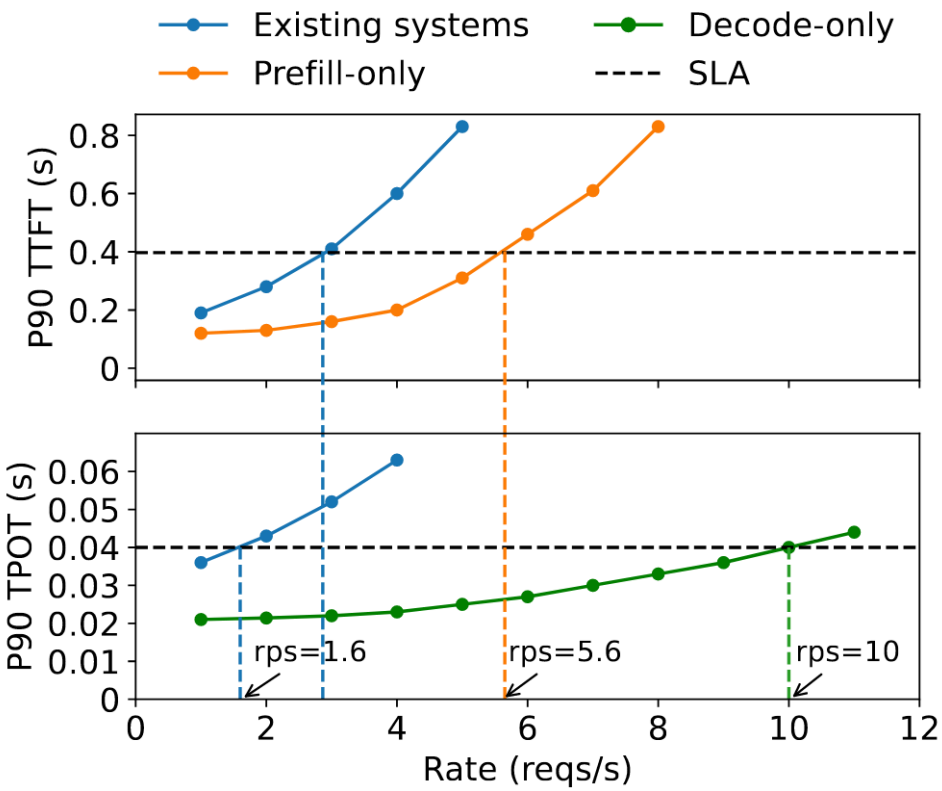


Figure 1: Performance when serving an LLM with 13B parameters under a synthetic workload with input length = 512 and output length = 64 on one NVIDIA 80GB A100. *Upper*: The P90 time-to-first-token (TTFT) latency comparing existing systems vs. a system serving only the prefill phase. *Down*: The P90 time-per-output-token (TPOT) latency comparing existing systems vs. a system serving only the decoding phase.

实验配置：

- 1张A100 80G的gpu
- 13B LLM，input_length = 512, output_length = 64

SLO（人为定义的系统性能达标要求）：

- P90 TTFT = 0.4s，人为定义在prefill阶段90%的请求的TTFT必须达到0.4s（上面那张图黑色虚线部分）
- P90 TPOT = 0.04s，人为定义在decode阶段90%的请求的TPOT必须达到0.04s（下面那张图的黑色需要部分）

现有系统（共置prefill 和 decoding）在单个 A100 GPU 上可实现的最大吞吐量（受到 TTFT 和 TPOT 要求中更严格的一项的限制）约为每秒 1.6 个请求 (rps)。

只做 prefill 请求的per GPU goodput为 5.6 rps（橙色），只做 decoding请求的per GPU goodput为 10 rps（绿色）。理想情况下，通过分配 2 个 GPU 进行预填充和 1 个 GPU 进行解码，我们可以有效地以 10 rps 的总体吞吐量（即每个 GPU 3.3 rps）为模型提供服务，这比现有系统高 2.1 倍。

结论：一张卡只做prefill或者只做decode的goodput，都要比它既做prefill又做decode的goodput要高

prefill/decode 分离的好处

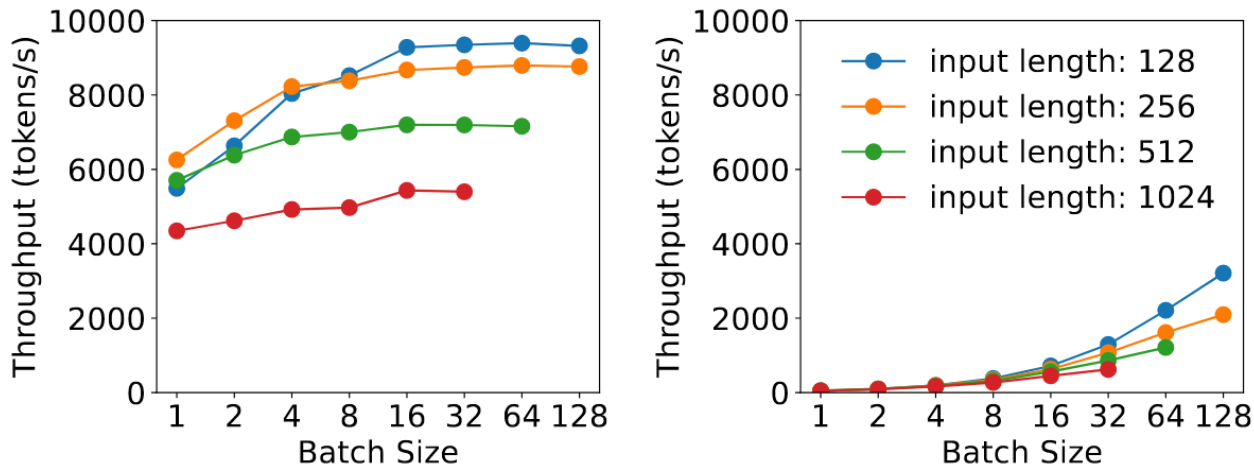
- 消除预填充解码干扰
- 允许根据 prefill 和 decoding 的不同特性制定不同的资源分配和模型并行策略，以满足其特定的延迟要求

分离架构下prefill/decode的优化

两阶段的不同特性

- **prefill阶段：拥有计算受限的性质（compute-bound）**，特别是在请求流量较大，用户的prompt也比较长的情况下。
- **decode阶段：拥有存储受限的性质（memory-bound）**，因为token by token的生成方式，decode阶段要频繁从存储中读取KV Cache，同时也意味着它需要尽可能保存KV cache。

Batching strategy



(a) Prefill phase (b) Decoding phase

Figure 3: Throughput for two phases with different batch sizes and input lengths when serving an LLM with 13B parameters.

- **prefill阶段：随着batch size的增加，吞吐量的增长趋势却趋于平缓**。这正是因为prefill阶段是compute-bound的，当batch中的总tokens数大于一个阈值的时候，prefill阶段产生compute-bound，使得吞吐量的增长趋势变缓（这里不考虑显存问题，因为在该实验中硬件显存足够这些不同batchsize和input_length的数据使用）。
- **decode阶段：随着batch size的增加，吞吐量的增长趋势越来越显著**。这是因为decode阶段是memory-bound的，即相比于计算，读写数据的时间要更多。所以在decode阶段中提升batch size吞吐量就上去了。

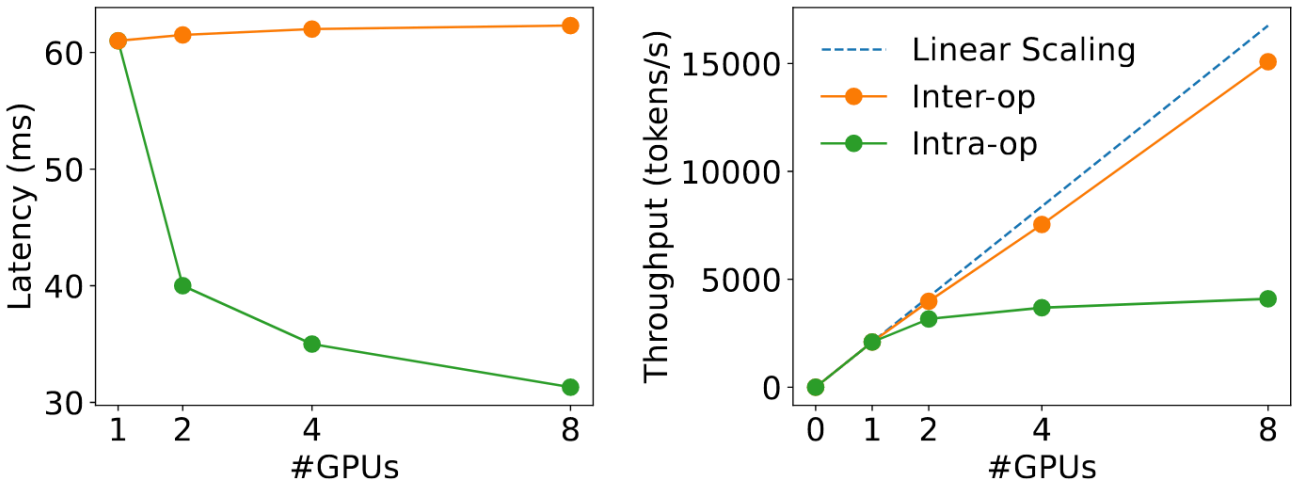


Figure 5: Decoding phase latency and throughput when serving a 13B LLM with batch size = 128 and input length = 256 under different parallel degrees.

Parallelism plan

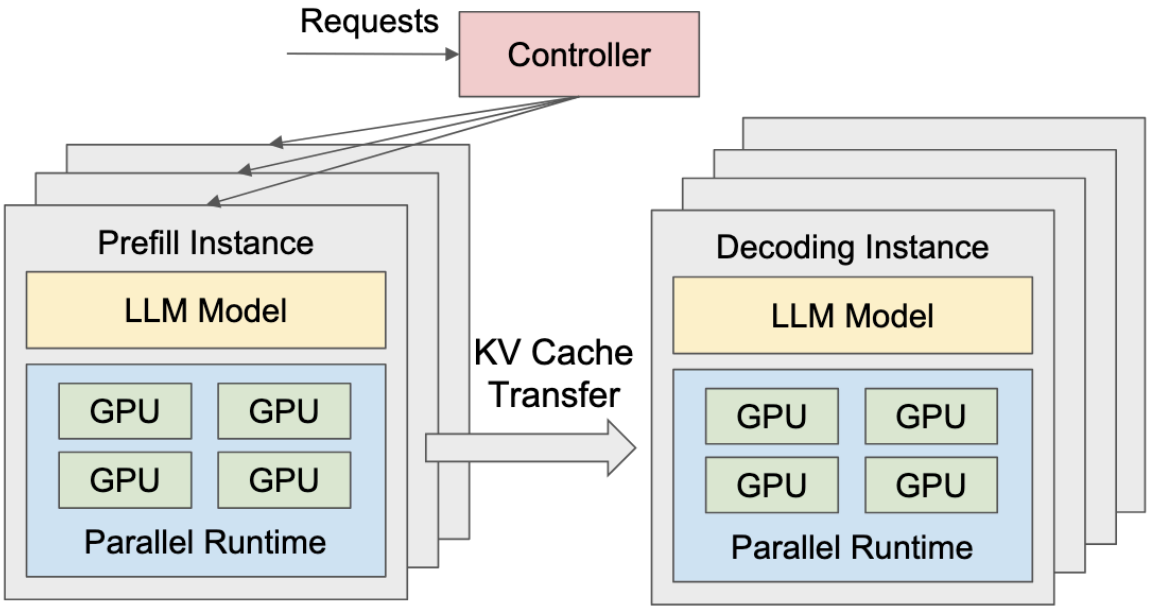


Figure 6: DistServe Runtime System Architecture

一个prefill/decode instance包含一个完整的模型副本，先考虑实例内的 tp/pp；再考虑复制多个实例（dp）。

prefill

单卡

假设队列中的请求服从排队论中的M/D/1模式：

- **M：请求的到达过程遵从柏松分布**。即请求的到达系统是相互独立的（不存在互相干扰）且单位时间内各请求到达系统的概率是等可能的。
- **D：假设所有请求做prefill的处理时间相同**。（请求输入长度为 512 tokens）
- 1：可以理解成只有1张gpu

单卡下每个请求的平均排队时间可以建模为：

$$Avg\_TTFT = D + \frac{RD^2}{2(1-RD)}. \quad (1)$$

D表示每个请求 prefill 的执行时间，R 表示请求到达率，后一项表示在队列中的等待时间。

2卡pp

$$Avg\_TTFT_{inter} = D_s + \frac{RD_m^2}{2(1-RD_m)} = D + \frac{RD^2}{4(2-RD)}. \quad (2)$$

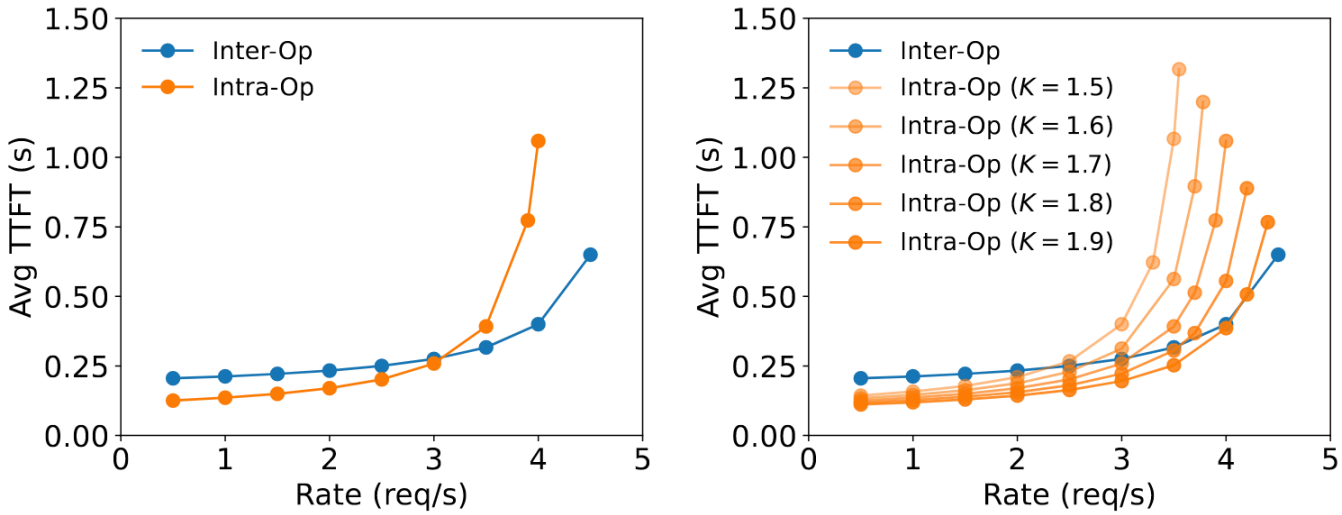
执行时间 $D_s = D$，等待时间$Dm = D/2$

2卡tp

$$Avg\_TTFT_{intra} = \frac{D}{K} + \frac{RD^2}{2K(K-RD)}. \quad (3)$$

K 表示加速系数（$1 < K < 2$），执行时间$D_s = D/K$

建模结果



(a) Real experiment results (b) Changing intra-op speedup

Figure 4: Average TTFT when serving an LLM with 66B parameters using different parallelism on two A100 GPUs.

对于prefill阶段，在rate较小时，更适合用tp；在rate较大时，更适合用pp。即prefill阶段在不同条件下，对并行方式有倾向性。

decoding

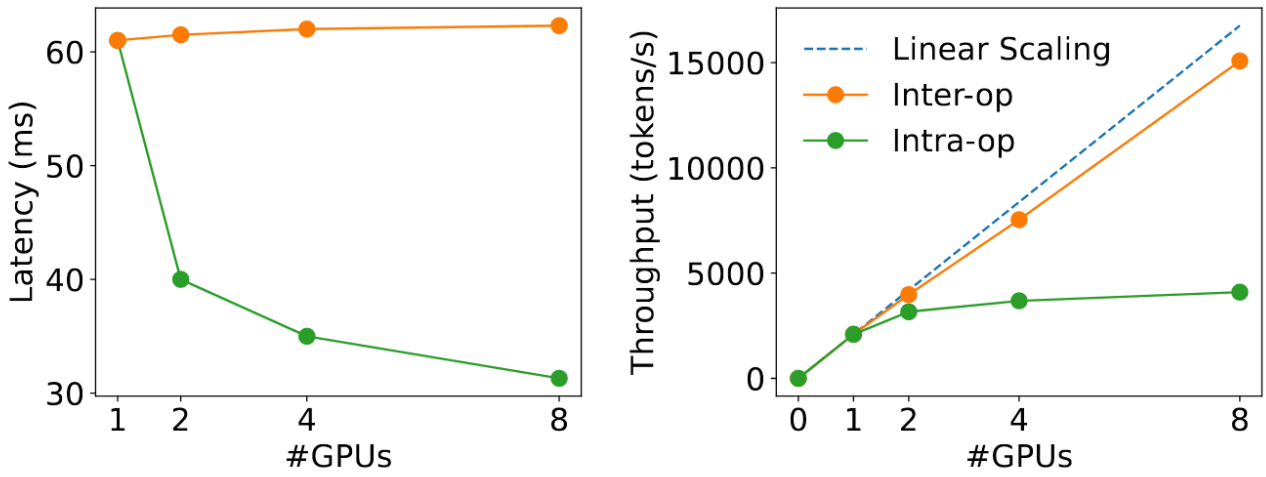


Figure 5: Decoding phase latency and throughput when serving a 13B LLM with batch size = 128 and input length = 256 under different parallel degrees.

Method

给定模型、工作负载特征、延迟要求和 SLO attainment，DistServe 需要确定以下要素（称作 placement）：

- prefill和decoding实例的并行策略（tp/pp）
- 确定 prefill 和 decoding 阶段要部署的实例数量（dp）
- 如何将它们放置到物理集群上。

Target：找到一个能够最大化每 GPU 吞吐量的placement

两种不同的硬件场景

- 集群中 node 之间带宽很高：忽略节点间的 KV cache传输时间
- 集群中 node 之间带宽有限：KV cache的传输时间不能被忽略

Placement for High Node-Affinity Cluster

Algorithm 1 High Node-Affinity Placement Algorithm

Input: LLM G, #node limit per-instance N, #GPU per-node M, GPU memory capacity C, workload W, traffic rate R.
Output: the placement $best\_plm$.

$config_p, config_d \leftarrow \emptyset, \emptyset$
for $intra\_op \in \{1, 2, ..., M\}$ **do**
  for $inter\_op \in \{1, 2, ..., \frac{N \times M}{intra\_op}\}$ **do**
    if $\frac{G.size}{inter\_op \times intra\_op} < C$ **then**
      $config \leftarrow (inter\_op, intra\_op)$
      $\hat{G} \leftarrow \text{parallel}(G, config)$
      $config.goodput \leftarrow \text{simu\_prefill}(\hat{G}, W)$
      if $\frac{config_p.goodput}{config_p.num\_gpus} < \frac{config.goodput}{config.num\_gpus}$ **then**
        $config_p \leftarrow config$
      $config.goodput \leftarrow \text{simu\_decode}(\hat{G}, W)$
      if $\frac{config_d.goodput}{config_d.num\_gpus} < \frac{config.goodput}{config.num\_gpus}$ **then**
        $config_d \leftarrow config$
$n, m \leftarrow \lceil \frac{R}{config_p.goodput} \rceil, \lceil \frac{R}{config_d.goodput} \rceil$
$best\_plm \leftarrow (n, config_p, m, config_d)$
return $best\_plm$

M表示一个 node 中的 GPU 数量

N表示一个实例最多能够占的 node 数量

系统的负载情况：与模型类型、input_length 有关，在真实场景中很难对系统的负载情况进行建模。因此可以通过模拟实验的方式，先收集一段时间的用户真实数据，然后用一个模型拟合出用户真实请求服从的分布，从这个分布中采样得到出一批数据作为系统的负载。运用采样的数据对并行策略profiling了。

算法的目标是找到使得 per gpu goodput 最大的 placement config，包括以下四个部分：

- `config_p`: prefill阶段最佳的并行方式（每个config_p装一套完整的模型，称为一个prefill实例）
- `config_d`: decode阶段最佳的并行方式（每个config_d装一套完整的模型，称为一个decode实例）

- `n`: prefill阶段的dp值，即要部署多少个prefill实例
- `m`：decode阶段的dp值，即要部署多少个decode实例

在配置 G 下，对于负载 W 能够达到 SLO 值的最大 rate 值就是该config对应的 goodput，通过网格搜索找到一个实例中最佳的 config_p 和 config_d。

再根据系统的流量确定 n 和 m 值。

Placement for Low Node-Affinity Cluster

Algorithm 2 Low Node-Affinity Placement Algorithm

Input: LLM G, #node limit per-instance N, #GPU per-node M, GPU memory capacity C, workload W, traffic rate R.
Output: the placement *best_plm*.

$config^* \leftarrow \emptyset$
for $inter\_op \in \{1, 2, ..., N\}$ **do**
  $\mathcal{P} \leftarrow \text{get\_intra\_node\_configs}(G, M, C, inter\_op)$
  for $P_p \in \mathcal{P}$ **do**
    for $P_d \in \mathcal{P}$ **do**
      if $P_p.num\_gpus + P_d.num\_gpus \leq M$ **then**
        $config \leftarrow (inter\_op, P_p, P_d)$
        $\hat{G}_p, \hat{G}_d \leftarrow \text{parallel}(G, config)$
        $config.goodput \leftarrow \text{simulate}(\hat{G}_p, \hat{G}_d, W)$
        if $\frac{config.^*goodput}{config.^*num\_gpus} < \frac{config.goodput}{config.num\_gpus}$ **then**
          $config^* \leftarrow config$
$n \leftarrow \lceil \frac{R}{config.^*goodput} \rceil$
$best\_plm \leftarrow (n, config^*)$
return *best_plm*

将 decode 和 prefill 放置在同一个 node 中，减少 KV cache 传输时间

要求 decode 和 prefill 的 pp 维度是一样的

实验设置

- **集群配置**：DistServe部署在一个包含4个节点和32个GPU的集群上，每个节点有8个NVIDIA SXM A100 80GB GPU，通过NVLINK连接，跨节点带宽为25Gbps。
- **模型选择**：使用OPT系列模型进行实验，设置为FP16精度。
- 工作负载和SLO设置
  - **ChatBot**：使用ShareGPT数据集，设置了不同的TTFT和TPOT SLO。
  - **Code Completion**：使用HumanEval数据集，设置严格的SLO。
  - **Summarization**：使用LongBench数据集，设置宽松的TTFT SLO和严格的TPOT SLO。

Application	Model Size	TTFT	TPOT	Dataset
Chatbot OPT-13B	26GB	0.25s	0.1s	ShareGPT [8]
Chatbot OPT-66B	132GB	2.5s	0.15s	ShareGPT [8]
Chatbot OPT-175B	350GB	4.0s	0.2s	ShareGPT [8]
Code Completion OPT-66B	132GB	0.125s	0.2s	HumanEval [14]
Summarization OPT-66B	132GB	15s	0.15s	LongBench [13]

Table 1: Workloads in evaluation and latency requirements.

- **SLO评估指标**：主要关注90%的SLO达成率。
- **基准对比**：与vLLM和DeepSpeed-MII进行对比，后者支持分块预填充，但无法消除长预填充作业带来的解码干扰。

实验目标是评估DistServe在不同工作负载和SLO要求下的性能。

End-to-end Experiments

根据Low Node-Affinity Placement 算法得到的并行策略

Model	Dataset	Prefill		Decoding	
		TP	PP	TP	PP
OPT-13B	ShareGPT	2	1	1	1
OPT-66B	ShareGPT	4	1	2	2
OPT-66B	LongBench	4	1	2	2
OPT-66B	HumanEval	4	1	2	2
OPT-175B	ShareGPT	3	3	4	3

Table 3: The parallelism strategies chosen by DistServe in the end-to-end experiments.

实验结果

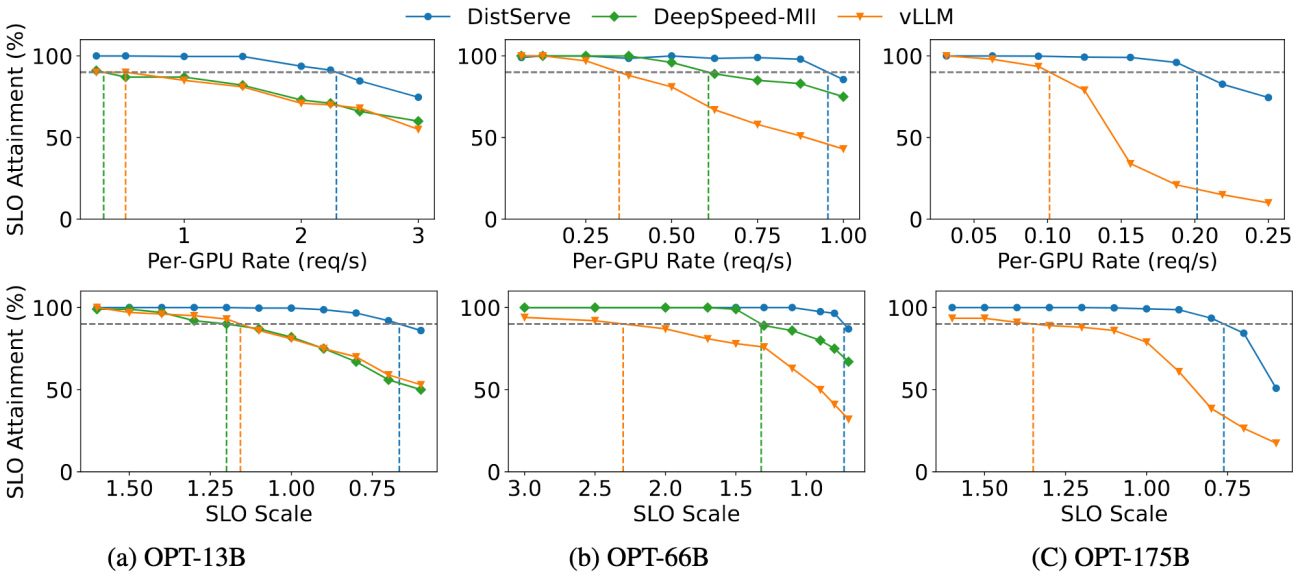


Figure 8: Chatbot application with OPT models on the ShareGPT dataset.

SLO Scale 表示的是对TTFT 和 TPOT 两个指标进行线性放缩程度，SLO Scale 越低表示 SLO 要求越严格。

Latency Breakdown

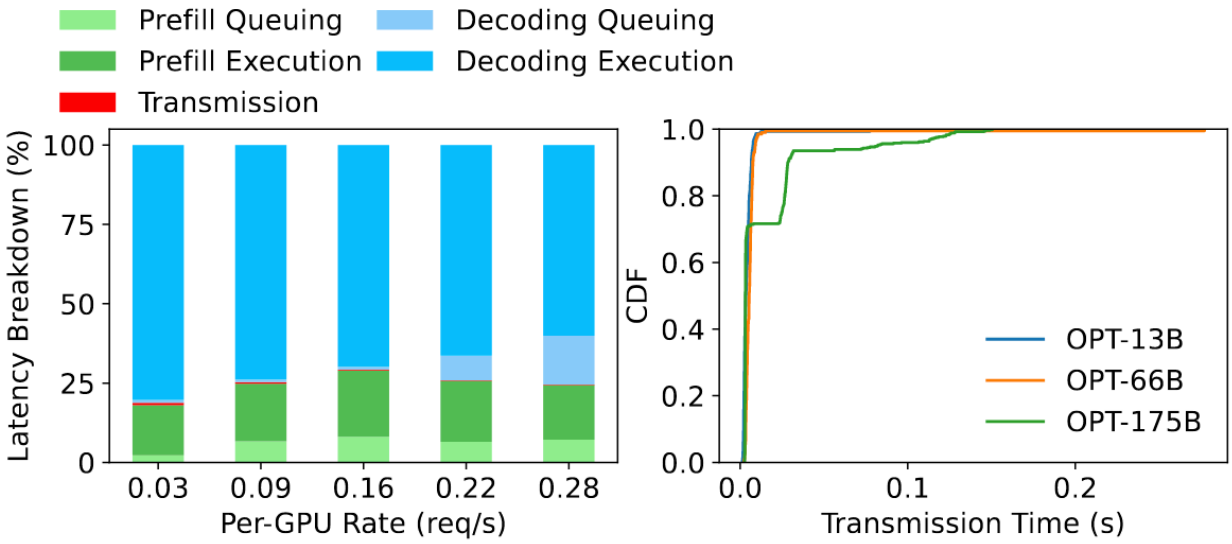


Figure 10: *Left:* Latency breakdown when serving OPT-175B on ShareGPT dataset with DistServe. *Right:* The CDF function of KV Cache transmission time for three OPT models.

由 DP 分离导致的KV Cache 传输的占比很低，占总延迟比例不到 0.1%。